

Spring MVC(Legacy)

프로젝트로 알아보는 Spring

프레임워크의 개념

- 프레임워크란
 - 아키텍처에 해당하는 골격 코드
 - 아키텍처 란
 - 전체 애플리케이션의 구조를 결정
 - 애플리케이션의 기본 아키텍처는 프레임워크가 제공, 나머지는 개발자가 담당
- 프레임워크의 장점
 - 빠른 구현 시간
 - 쉬운 관리
 - 개발자 역량 획일화
 - 검증된 아키텍처의 재사용과 일관성 유지

스프링 프레임워크

- 로드 존슨(Rod Johnson)이 2002년에 출판한 저서 Expert One-on-One J2EE Design and Development에서 선보인 예제 소스 코드에서 시작하여 현재까지 발전된 자바 기반의 웹 프레임워크.
- 한국 전자정부 표준 프레임워크의 기반 기술이며, 한국 정보화 진흥원에서 공공기관의 웹 서비스 제공 시에 스프링 프레임워크를 권장하고 있음.

스프링 프레임워크

- POJO(Plain Old Java Object) 방식
 - '오래된 방식의 간단한 자바 오브젝트'라는 뜻. Java EE(EJB) 등의 중량 프레임워크들을 사용하게 되면서 해당 프레임워크에 종속된 "무거운" 객체를 만들게 된 것에 반발해서 사용되게 된 용어. 특정 자바 모델이나 기능, 프레임워크 등을 따르지 않은 자바 오브젝트를 지칭.
- AOP(Aspect Oriented Programming)
 - '관점 지향 프로그래밍'. 로깅, 트랜잭션, 보안 등 여러 부분에서 공통적으로 사용되는 코드(기능)를 분리하여 관리하는 프로그래밍 방식.
- DI(Dependency Injection)
 - '의존성 주입'. 객체 간의 의존 관계를 소스 코드 내부에서 처리하지 않고, 외부 설정으로 정의되는 방식. 소소의 재사용성과 객체 간의 결합도를 낮출 수 있음. 스프링 프레임워크가 DI를 처리.

스프링 프레임워크

- IoC(Inversion of Control)
 - '제어의 역전'. 개발자가 작성한 코드가 외부 라이브러리를 사용하는 방식(전통적인 방식)이 아니라 외부 라이브러리(프레임워크)가 개발자의 코드를 필요에 따라 사용하는 방식.
- Lifecycle 관리
 - 자바 객체의 생성, 소멸을 프레임워크가 관리.

스프링 프레임워크

- 핵심 개념

- DI(Dependency Injection, 의존성 주입)

- 일체형

- HAS-A 관계
 - A가 B를 생성하는 관계.

```
class Car { Car(){ new Engine(); } }  
...  
Car car = new Car();
```

- 분리/부착형(DI)

- A 객체가 B 객체를 사용하는 관계

```
Engine eg = new Engine();  
Car car = new Car();  
Car.setEngine(eg);
```

스프링 프레임워크

- 핵심 개념

- DI 종류

- Setter Injection

```
B b = new B();  
A a = new A();  
a.setB(b);
```

- Constructor Injection

```
B b = new B();  
A a = new A(b);
```

- 스프링에서의 DI

- 부품들을 생성하고, 제품을 조립해주는 공정 과정을 대신해 주는 라이브러리

스프링 프레임워크

- 핵심 개념

- DI 구현

- 객체의 생성과 도킹에 대한 내용이 소스 코드 상에 있는 것이 아닌 별도의 XML 설정 파일에 분리하여 존재.
 - JAVA소스 컴파일 없이 XML 변경만으로 내용 변경 가능

config.xml

```
<bean id="record" class="di.SprRecord"></bean> // 빈 객체 생성
<bean id="view" class="di.SprRecordView"> // 빈 객체 생성
    <property name="record" ref="record"></property> // setRecord() 호출
</bean>
```

Java의 어느 클래스

```
// XML을 파싱하여 컨테이너에 담는 작업
ApplicationContext ctx = new ClassPathXmlApplicationContext("config.xml");
RecordView = (RecordView) ctx.getBean("view");
```


스프링 프레임워크

- 핵심 개념

- IoC(Inversion of Control, 제어의 역전)
 - 외부(컨테이너)에서 제어를 함
 - 객체 간의 낮은 결합도 유지
 - 프레임워크에 제어의 권한을 넘김으로써 클라이언트 코드가 신경 써야 할 것을 줄이는 전략
 - 컨테이너가 객체 생성, 객체 사이의 의존관계 처리
 - 기본적인 완제품 제작 순서와는 다르게 작은 부품부터 큰 부품으로, 제품을 만드는 순서가 역순
 - 이러한 일련의 작업을 스프링은 컨테이너라는 곳에 담아서 처리 -> IoC 컨테이너

스프링 프레임워크

- 핵심 개념

- IoC 컨테이너

- 빈(Bean)

- 스프링이 제어권을 가지고 직접 만들고 관계를 부여하는 오브젝트

- 빈 팩토리(Bean Factory)

- 빈(오브젝트)의 생성과 관계 설정 제어를 담당하는 IoC 오브젝트
 - 애플리케이션 컨텍스트(application context)라고도 함

- 애플리케이션 컨텍스트(IoC 컨테이너 or 스프링 컨테이너)

- DI를 위한 빈 팩토리에 엔터프라이즈 애플리케이션을 개발하는 데 필요한 여러 가지 컨테이너 기능을 추가한 것

- 설정정보/설정 메타정보

- 구성정보 or 형상정보 (XML)

- 스프링 컨테이너(IoC 컨테이너)

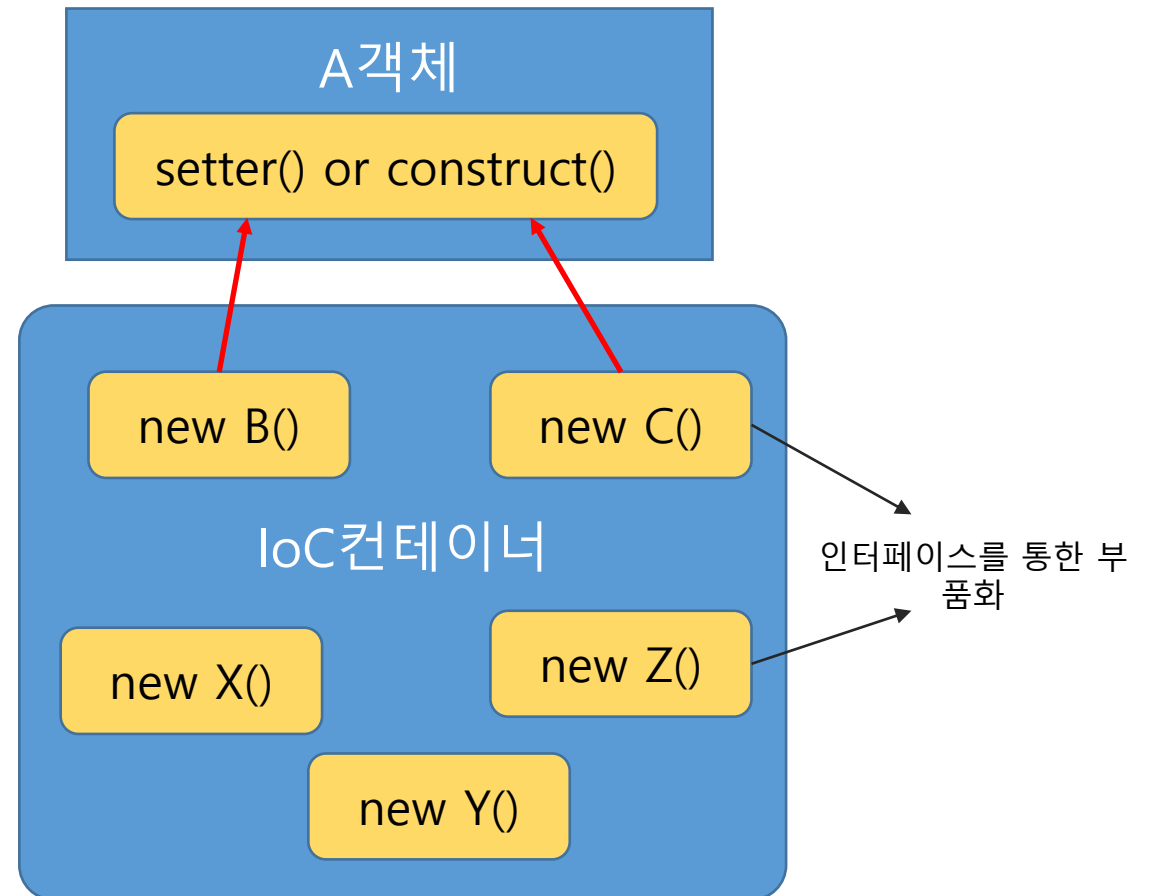
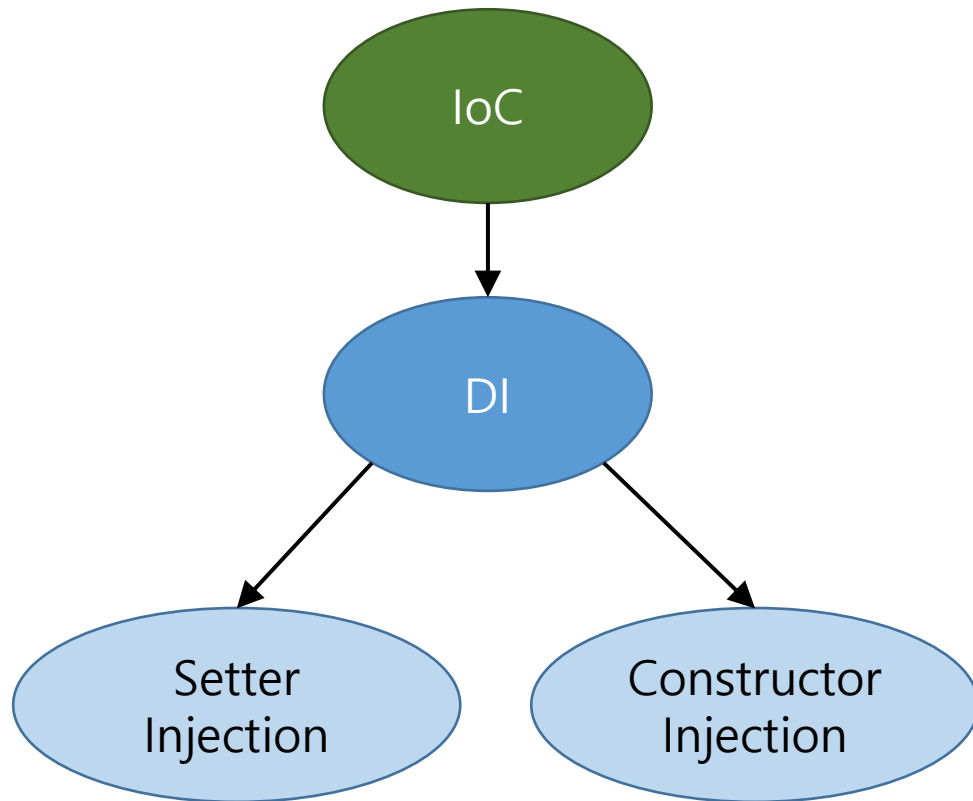
- IoC 방식으로 빈을 관리한다는 의미에서 애플리케이션 컨텍스트나 빈 팩토리를 의미

• 스프링 컨테이너 = IoC 컨테이너 = 애플리케이션 컨텍스트 = 빈 팩토리

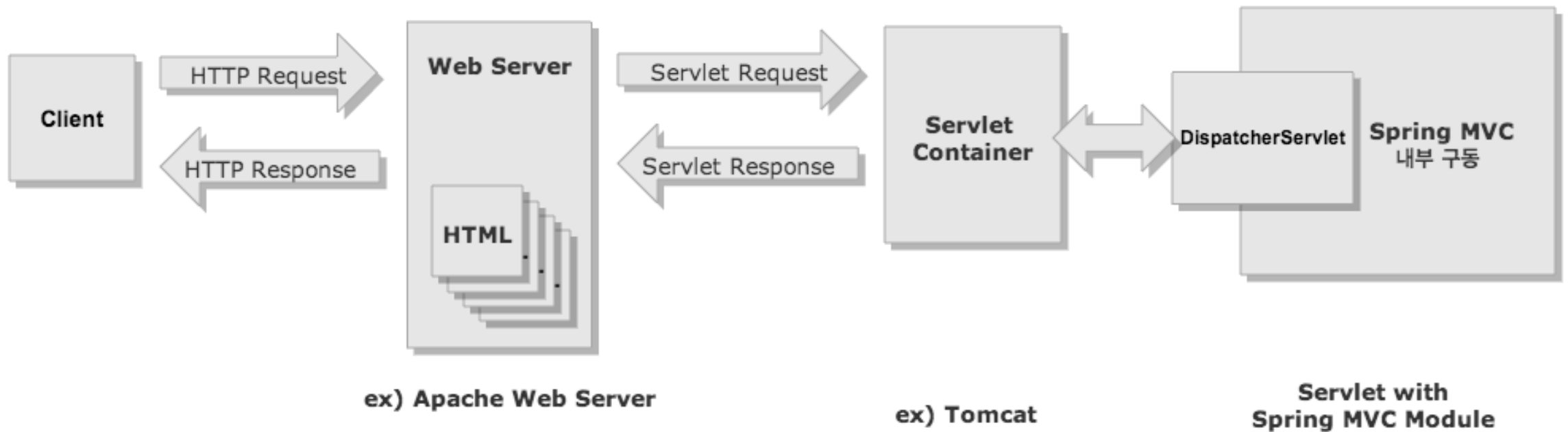
스프링 프레임워크

- 스프링의 DI와 IoC

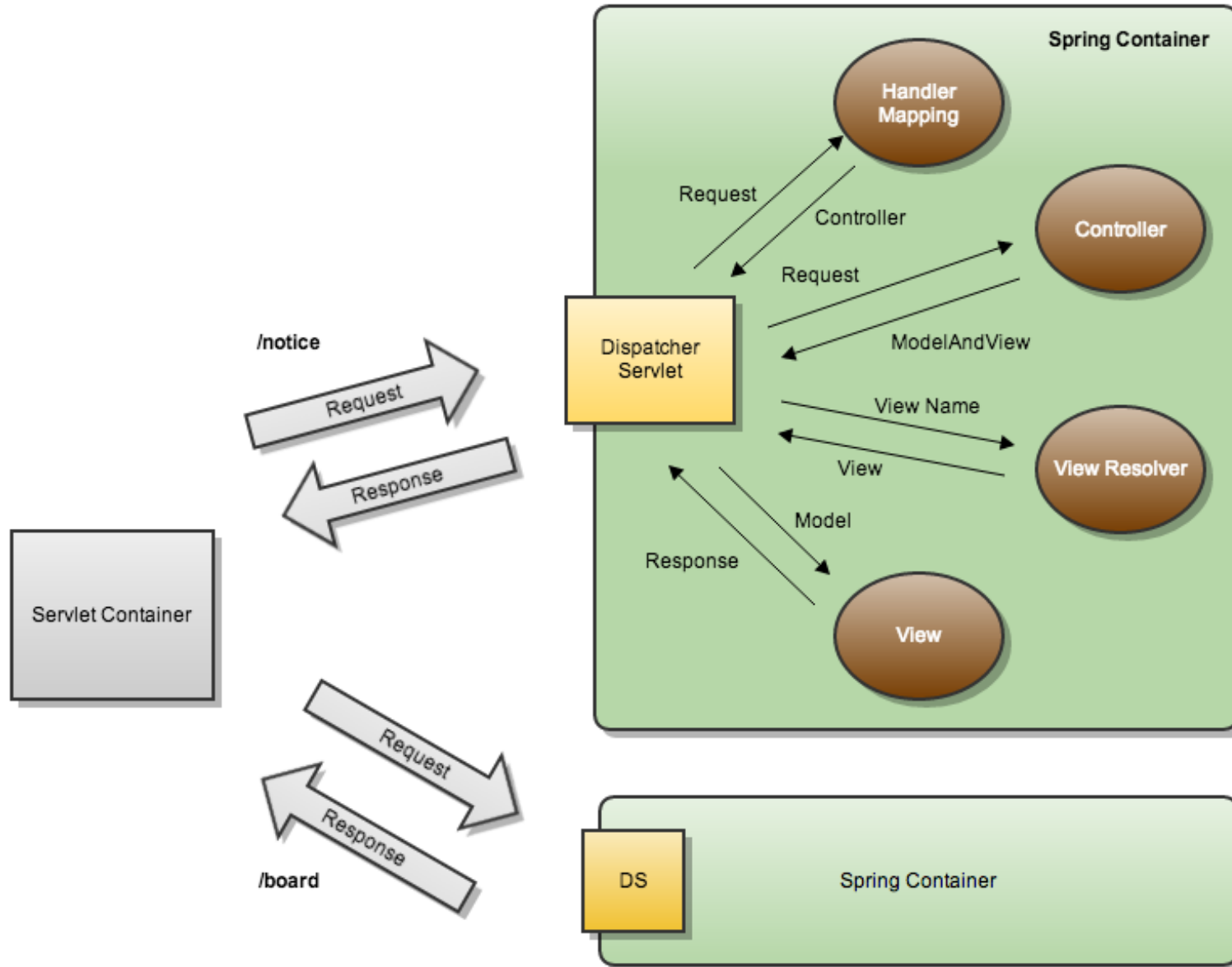
- 각 class 사이의 의존관계를 빈 설정 정보를 바탕으로 container가 자동적으로 연결



Spring MVC 웹서버 구동 방식



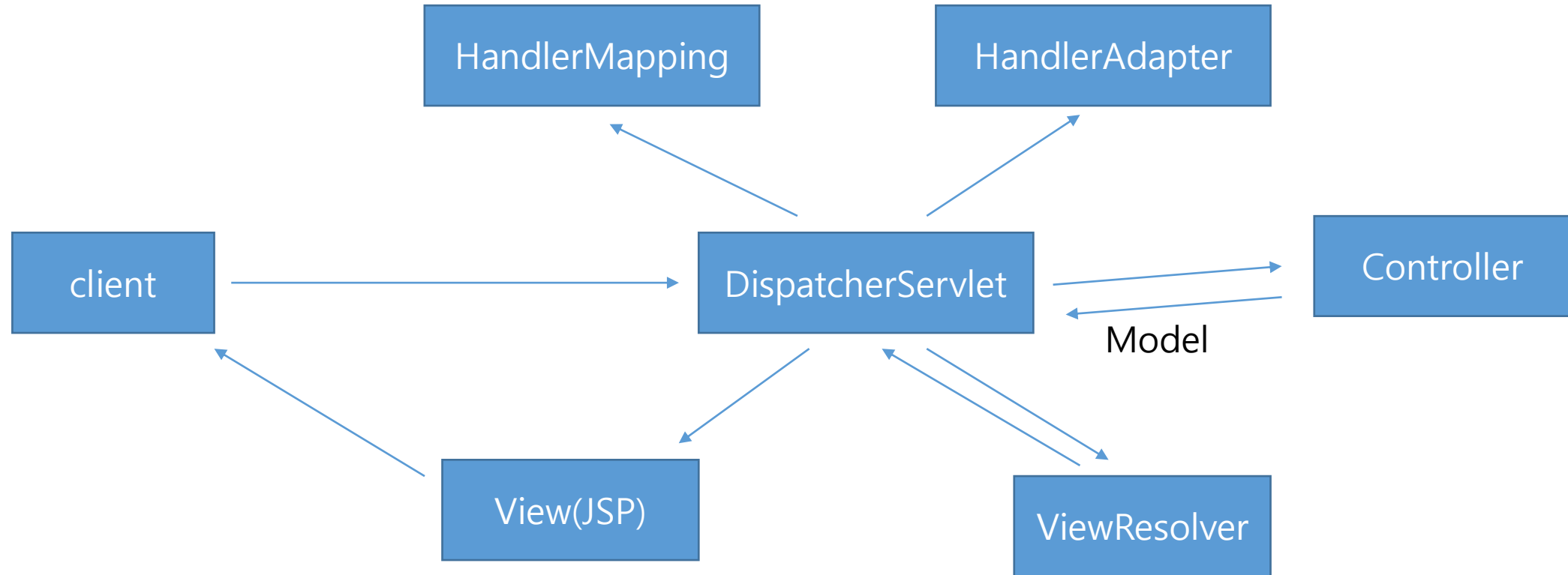
Spring MVC 웹서버 구동 방식



Spring MVC 웹서버 구동 방식

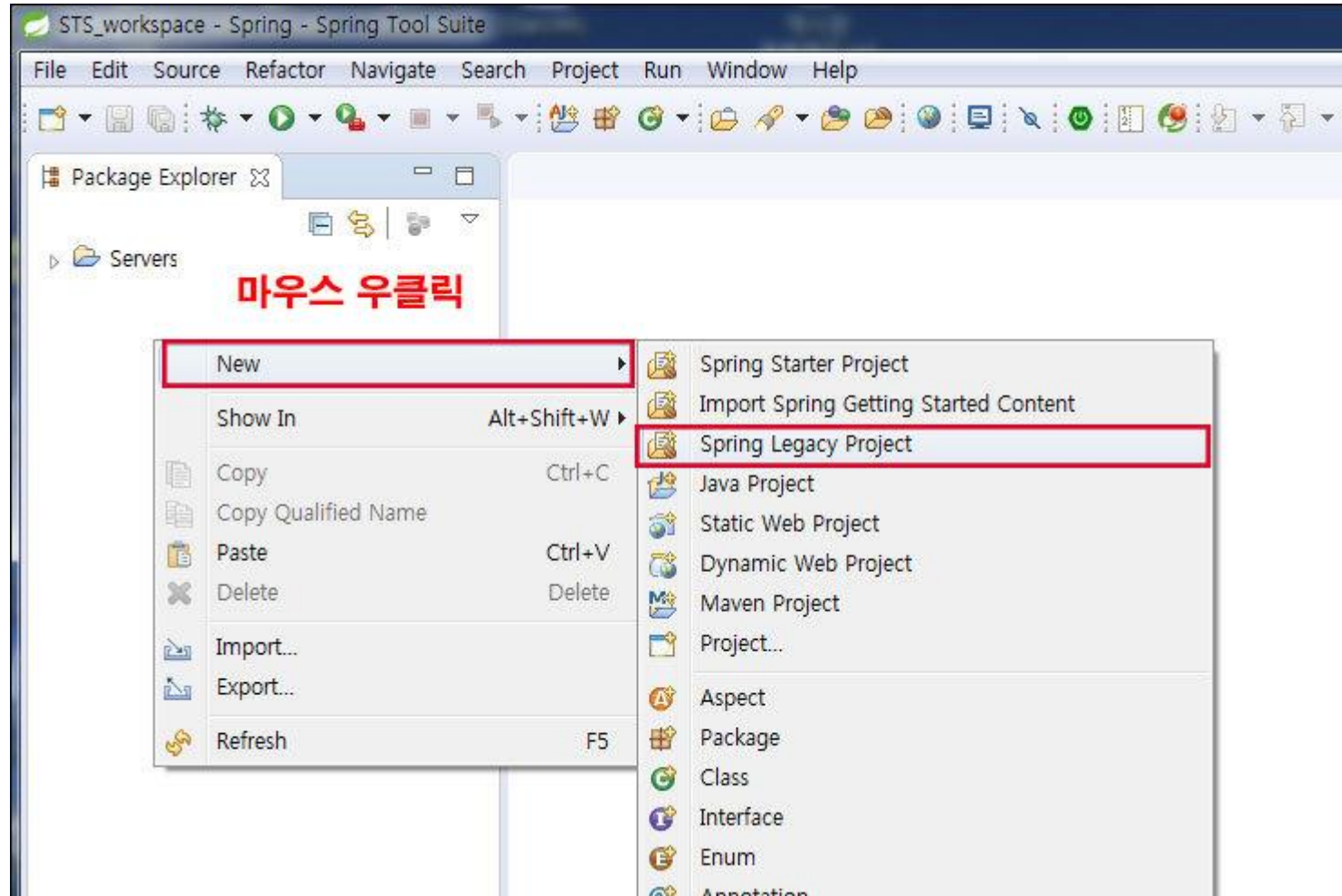
- Spring MVC module 내부의 작동
 - 서블릿이 요청을 수신(앞에서 Servlet Container가 적절한 서블릿으로 보낸준 것)
 - Handler Mapping을 통해서 요청을 처리할 Controller를 검색
 - 해당 Controller로 요청을 전송
 - Controller는 처리 결과를 Model로 반환
 - 반환 받은 View Name으로 View Resolver를 통해 View를 검색
 - 찾은 View 파일에 Controller가 만들었던 Model을 주어 View를 완성
 - 완성된 View를 Response가 Client로 전달
- Spring Container(=Application Container) 의 역할
 - Singleton의 bean들을 관리(Handler Mapper, Controller, View Resolver, View) : 요청마다 매번 새로 객체를 만드는 것 보다 Singleton으로 하나 만들어 두고 재사용
 - 개발자가 직접 Dispatcher Servlet과 각 bean들 사이의 의존성을 명시적으로 코드로 나타내지 않아도 DI로 각 컴포넌트 사이의 연결 생성

스프링 MVC 동작

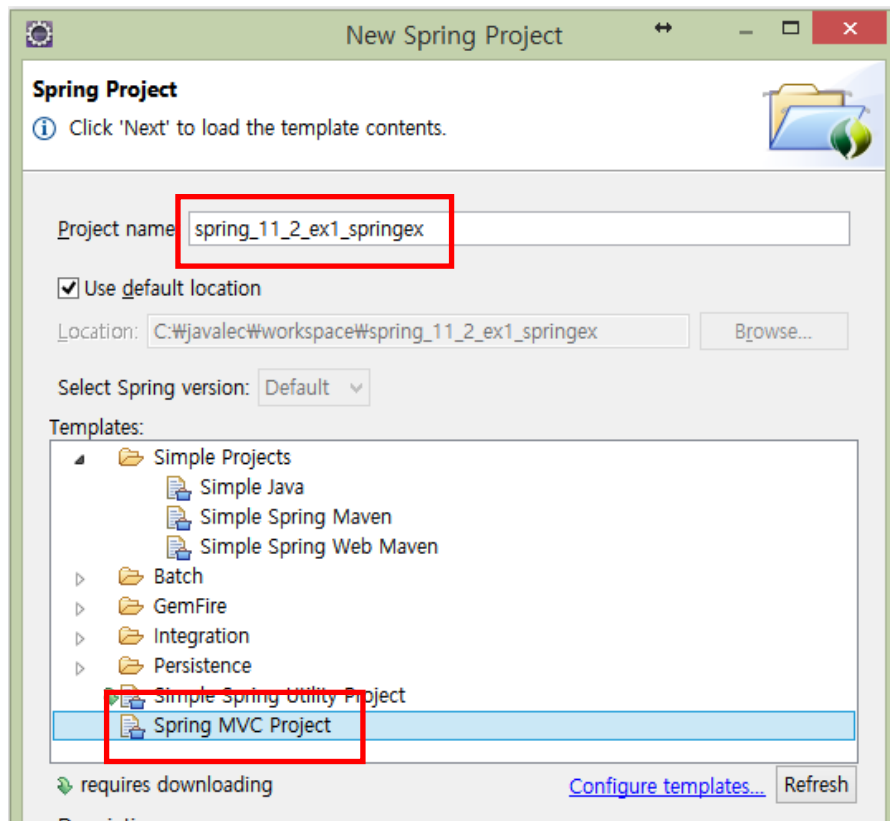


스프링 MVC 구조 살펴보기

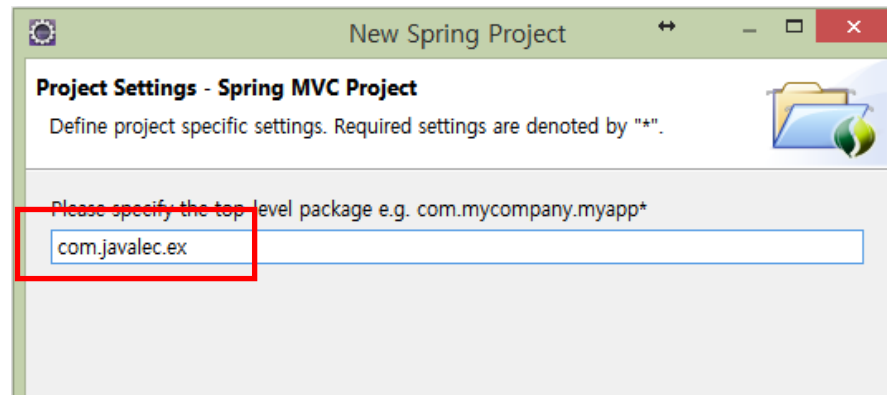
- 우선 스프링 MVC 프로젝트 생성



스프링 MVC 구조 살펴보기



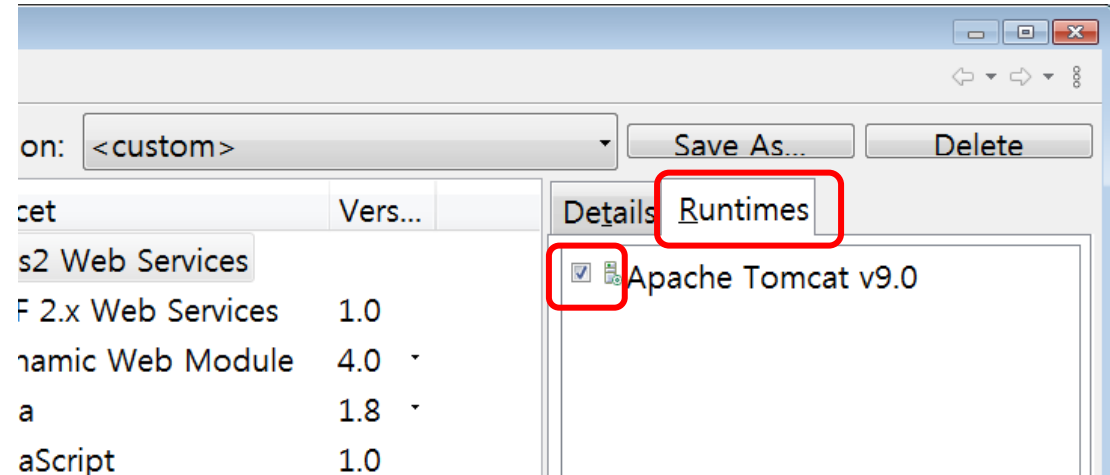
Project 이름 및 Spring MVC Project 설정



기본 패키지 설정

프로젝트 자바 및 웹 모듈 버전 변경

- Properties 변경
 - [프로젝트] 우클릭 > Properties
 - Project Facets >
 - Dynamic Web Module 2.5 -> 4.0
 - Java 1.6 -> 11
 - 대화창 우측 화면에 Runtimes 클릭 -> Apache Tomcat 체크
 - [Apply]



Spring 라이브러리 설정

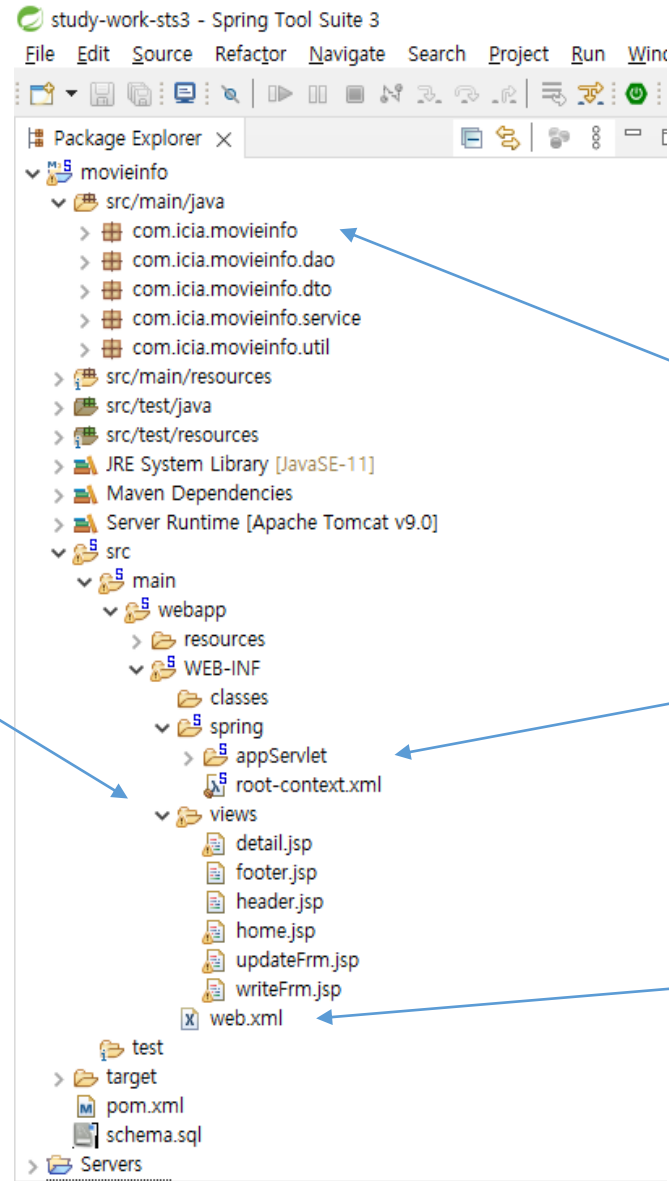
- 라이브러리 버전 수정(pom.xml)
 - 라이브러리 버전 확인 사이트 <https://mvnrepository.com/> (Maven Repository)
 - java-version : 11
 - springframework-version : 5.3.20
 - aspectj-version : 1.9.6
 - slf4j-version : 2.0.9
 - servlet-api -> javax.servlet-api version : 3.1.0
 - maven-compiler-plugin configuration의 source와 target : 11 (version 3.8.1)
 - 파일을 저장하면 자동으로 라이브러리 갱신
 - [프로젝트] 우클릭 > Maven > Update Protect... 실행
- 오류가 있을 경우(최종)
 - [프로젝트] 우클릭 > Maven > Disable Maven Nature 실행
 - [프로젝트] 우클릭 > Configure > Convert to Maven Project 실행
- 이 후 DB 연결, MyBatis, Secure, 파일업로드 관련 라이브러리 설정

한글 설정

- web.xml(그대로 입력)

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

스프링 MVC 구조 살펴보기



컨트롤러

Dispatcher에서 전달된 요청을 처리

servlet-context.xml

스프링 컨테이너 설정 파일

DispatcherServlet

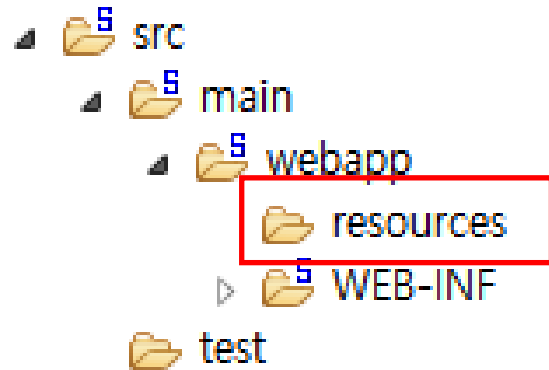
- 1) 클라이언트의 요청을 최초 받아
- 2) 컨트롤러에게 전달

web.xml

- 1) DispatcherServlet 서블릿 맵핑
- 2) 스프링 설정 파일 위치 정의

스프링 MVC 구조 살펴보기

- webapp/resources 폴더



```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

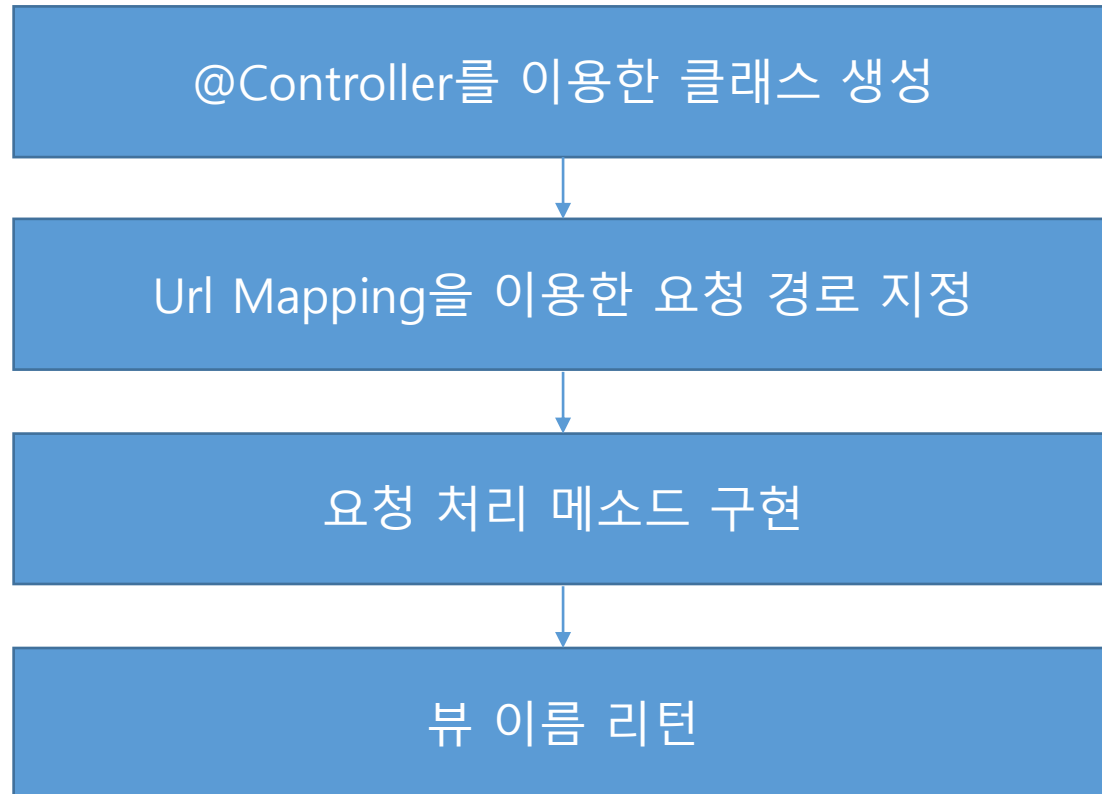
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

web.xml

```
<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />
```

컨트롤러 클래스 제작

- 최초 클라이언트로부터 요청이 들어왔을 때, 컨트롤러로 진입
- 컨트롤러는 요청에 대한 작업을 한 후 View쪽으로 데이터를 전달
- 컨트롤러 클래스 제작 순서



```
@Controller  
public class HomeController {
```

요청 처리 메소드 제작

- 클라이언트의 요청을 처리할 메소드

```
@RequestMapping("/board/view")  
public String view() {
```

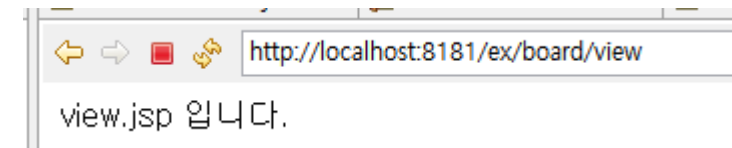
요청 경로(path)

```
    return "board/view";
```

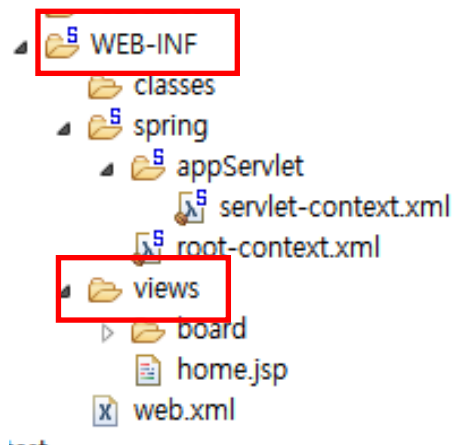
뷰페이지 이름

```
}
```

결과 화면



- 뷰페이지 이름 생성(조합) 방법



뷰페이지 이름 = prefix + 요청처리 메소드 반환값 + suffix

```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF  
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <beans:property name="prefix" value="/WEB-INF/views/" />  
    <beans:property name="suffix" value=".jsp" />  
</beans:bean>
```


뷰에 데이터 전달

- Model 클래스를 이용한 데이터 전달
 - 컨트롤러에서 로직 수행 후 뷰 페이지를 반환
 - 이때 뷰에서 사용하게 될 데이터를 객체로 전달

```
@RequestMapping("/board/content")  
public String content(Model model) {
```

Model 객체를 파라미터로 받음

```
    model.addAttribute("id", 30);  
    return "board/content";
```

Model 객체에 데이터를 담음

```
}
```

```
</head>
```

```
<body>
```

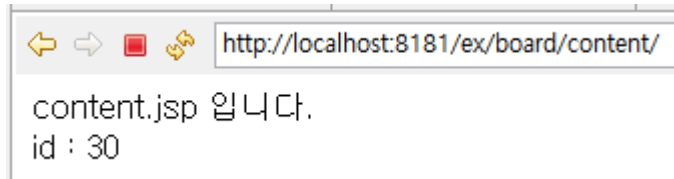
```
    content.jsp 입니다. <br />
```

```
    id : ${id}
```

```
</body>
```

```
</html>
```

컨트롤러에서 전달 받은
Model객체의 속성을 이용함.



뷰에 데이터 전달

- ModelAndView 클래스를 이용한 데이터 전달

```
@RequestMapping("/board/reply")  
public ModelAndView reply() {
```

```
    ModelAndView mv = new ModelAndView();  
    mv.addObject("id", 30);  
    mv.setViewName("board/reply");
```

```
    return mv;  
}
```

ModelAndView 객체 생성

Model 객체에 데이터를 담음

뷰이름 설정



http://localhost:8181/ex/board/reply

reply.jsp 입니다.
id : 30

클래스에 @RequestMapping 적용

- @RequestMapping 어노테이션을 클래스에 적용하여 요청 경로 획득

```
7 @Controller
8 @RequestMapping("/board")
9 public class HomeController {
10
```

클래스에 @RequestMapping 적용

/board



```
@RequestMapping("/write")
public String write(Model model) {

    model.addAttribute("id", 30);

    return "board/write";
}
```

메소드에 @RequestMapping 적용

/write



조합된 요청 경로 : /board/write

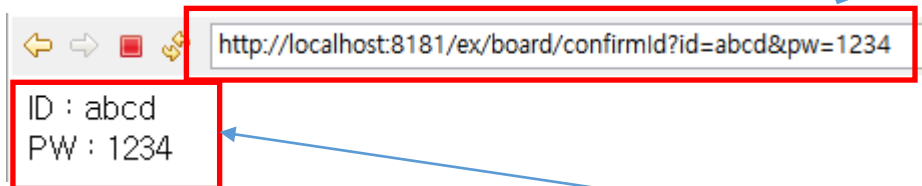
http://localhost:8181/ex/board/write

write.jsp 파일 입니다.

Form 데이터 전송

- HttpServletRequest 클래스를 이용해서 데이터 전송 방법

```
@RequestMapping("board/confirmId")  
public String confirmId(HttpServletRequest httpRequest, Model model) {  
    String id = httpRequest.getParameter("id");  
    String pw = httpRequest.getParameter("pw");  
    model.addAttribute("id", id);  
    model.addAttribute("pw", pw);  
    return "board/confirmId";  
}
```



```
</head>  
<body>  
    ID : ${id} <br />  
    PW : ${pw}  
</body>  
</html>
```

Form 데이터 전송

- @RequestParam 어노테이션을 이용한 데이터 전송 방법

```
@RequestMapping("board/checkId")  
public String checkId(@RequestParam("id") String id, @RequestParam("pw") int pw, Model model) {  
    model.addAttribute("identify", id);  
    model.addAttribute("password", pw);  
    return "board/checkId";  
}
```

http://localhost:8181/ex/board/checkId?id=abcdefg&pw=123456

ID : abcdefg
PW : 123456

```
</head>  
<body>  
    ID : ${identify} <br />  
    PW : ${password}  
</body>  
</html>
```

Form 데이터 전송

- 데이터(커맨드) 객체를 이용하여 데이터 많을 경우 간단하게 사용

기존 방법 : 다소 코드양이 많다.

```
@RequestMapping("/member/join")
public String joinData(@RequestParam("name") String name, @RequestParam("id") String id,
    @RequestParam("pw") String pw, @RequestParam("email") String email, Model model) {

    Member member = new Member();
    member.setName(name);
    member.setId(id);
    member.setPw(pw);
    member.setEmail(email);

    model.addAttribute("memberInfo", member);

    return "member/join";
}
```

개선 방법 : 코드양이 적다.

```
@RequestMapping("/member/join")
public String joinData(Member member)

    return "member/join";
}
```

http://localhost:8181/ex/member/join?name=홍길동&id=abc&pw=123&email=abc@abc.com

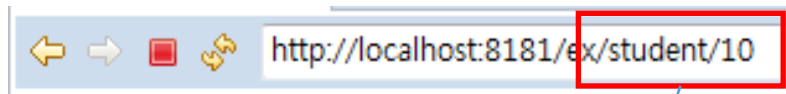
이름 : 홍길동
아이디 : abc
비밀번호 : 123
메일 : abc@abc.com

http://localhost:8181/ex/member/join?name=홍길동&id=abc&pw=123&email=abc@abc.com

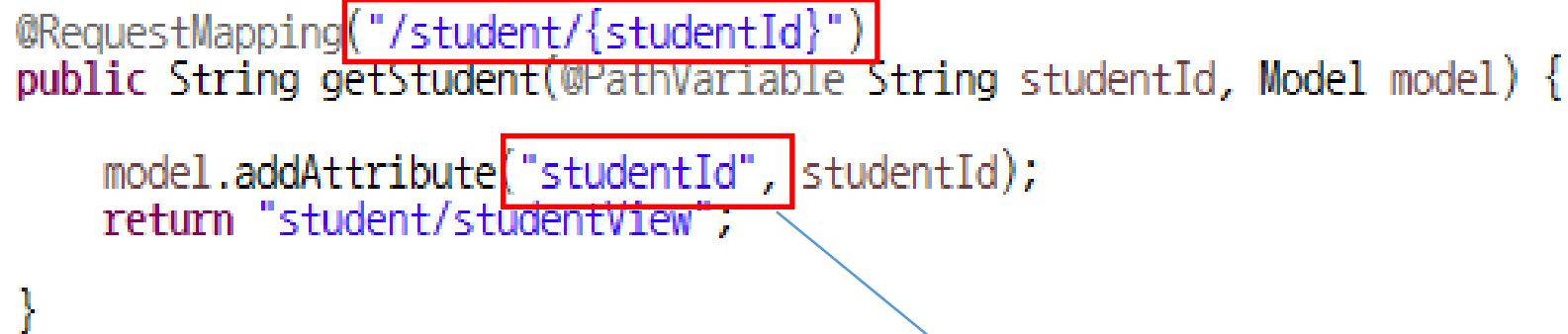
이름 : 홍길동
아이디 : abc
비밀번호 : 123
메일 : abc@abc.com

Form 데이터 전송

- @PathVariable 어노테이션을 이용하면 경로(path)에 변수를 넣어 요청메소드에서 파라미터로 이용

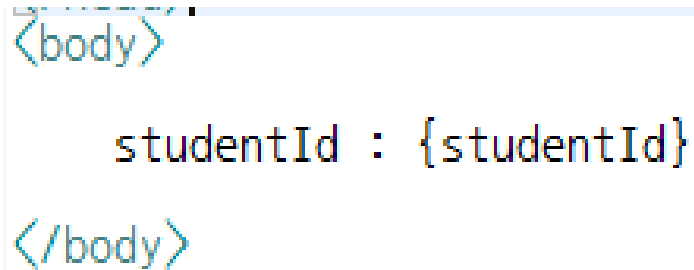


http://localhost:8181/ex/student/10



```
@RequestMapping("/student/{studentId}")
public String getStudent(@PathVariable String studentId, Model model) {

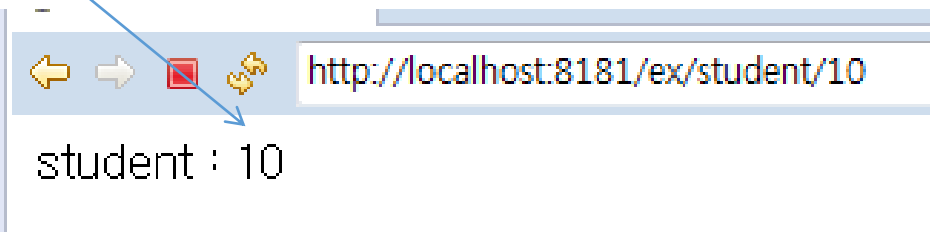
    model.addAttribute("studentId", studentId);
    return "student/studentView";
}
```



```
<body>

    studentId : {studentId}

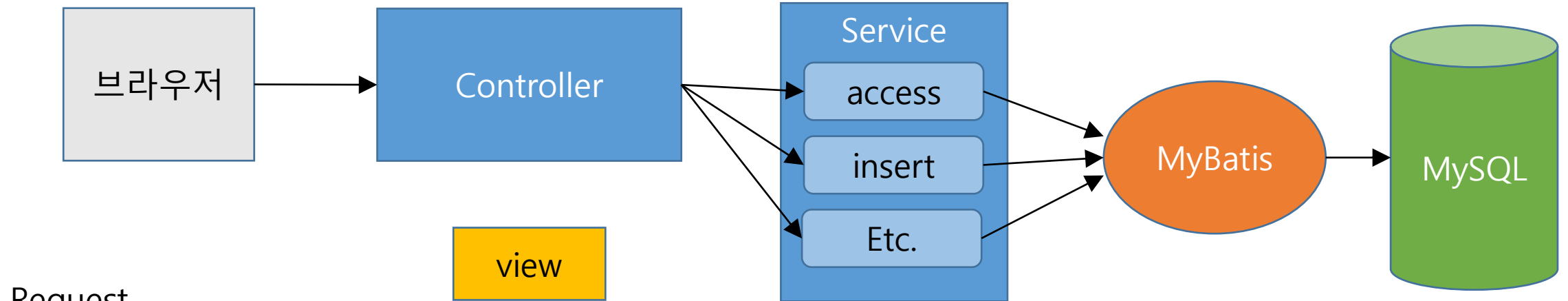
</body>
```



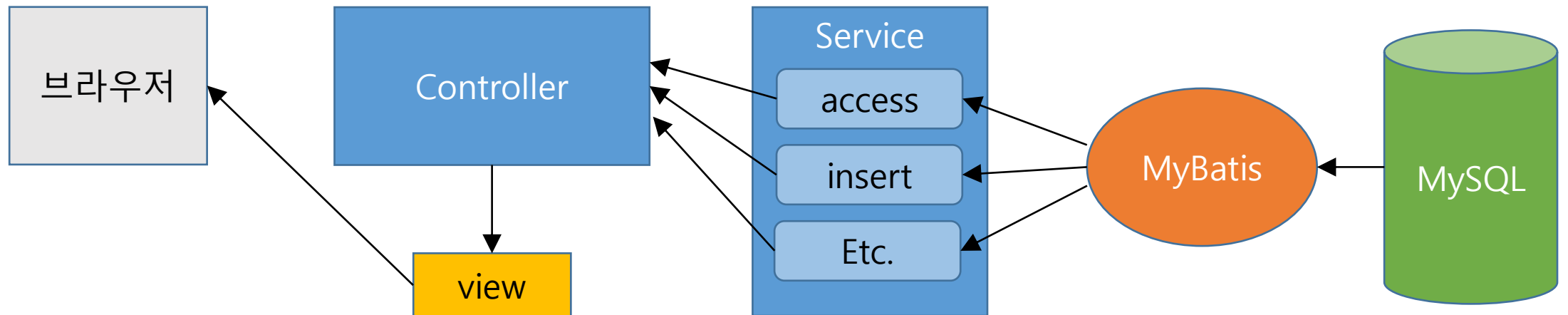
http://localhost:8181/ex/student/10

student : 10

Spring MVC 프로젝트 구조



Response



Spring MVC

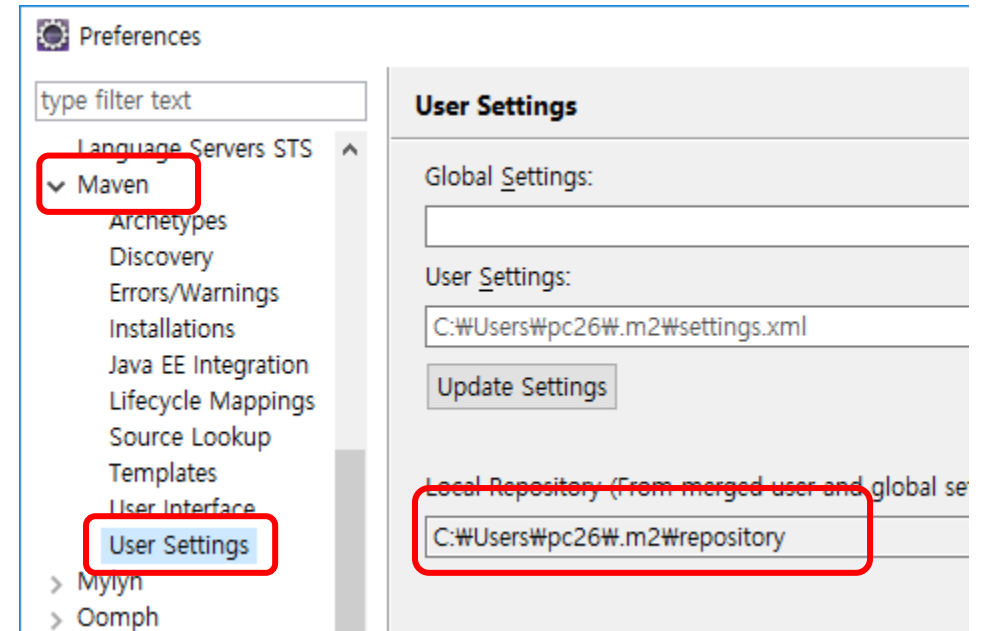
- Model(dto class)
 - Member class : 회원 정보
 - Board class : 게시글 정보
 - Bfile class : 첨부파일 정보
 - Reply class : 댓글 정보
- View(jsp)
 - home.jsp : 로그인 페이지
 - joinFrm.jsp : 회원 가입 페이지
 - boardList.jsp : 게시판 목록 페이지
 - writeFrm.jsp : 글쓰기 페이지
 - boardContent.jsp : 글 내용 보기 페이지(삭제, 댓글 포함) – Ajax 사용

Spring MVC

- Controller
 - HomeController : 회원 처리용(로그인/아웃, 회원 가입)
 - BoardController : 게시판 처리용(목록, 글쓰기, 내용 보기, 글삭제 등)
- Service
 - MemberManagement : 회원 관련 처리용 서비스 객체
 - BoardManagement : 게시판 관련 처리용 서비스 객체

Database 설정

- pom.xml에 라이브러리 설정
 - 설정 위치는 servlet과 test 사이(위치는 큰 의미 없음)
 - MyBatis, DBCP(DB Connection Pool), Spring jdbc, mysql connector 설정
 - Maven Repository에서 많이 사용하는 버전 선택
- Maven Repository
 - Artifact(라이브러리)들의 저장소로 로컬 및 원격 repository로 구성
 - 프로젝트의 pom.xml에서 선언한 dependency들의 artifact을 저장소로부터 불러와서 사용
 - Maven 설치 시 로컬에 Maven artifact들을 저장하고 관리하는 repository가 자동으로 구성
 - 로컬 위치 -> 그림



DB 설정 방법

- pom.xml에 라이브러리 설정
 - Spring jdbc

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.3.0</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
```

DI를 위한 어노테이션 Autowired

- 서비스 객체에 DAO 멤버변수 선언 후 사용
 - @Autowired 어노테이션을 사용하여 bean과 자동 연동

```
@Autowired  
private MemberDao mDao;
```

- 멤버 변수 이름은 bean의 id와 동일해야 함

Database 설정

- DBCP(DB Connection Pool)
 - 일반적인 DB 처리 과정
 - DB 서버 접속을 위해 JDBC 드라이버를 로드
 - DB 접속 정보와 DriverManager.getConnection() Method로 DB Connection 객체 획득
 - Connection 객체로 부터 쿼리를 수행하기 위한 Statement 객체 획득
 - 쿼리를 수행하여 그 결과를 받아서 데이터 처리
 - 처리가 완료되면 처리에 사용된 리소스들을 close하여 반환
 - 가장 느린 부분은 웹 서버에서 물리적으로 DB서버에 최초로 연결되어 Connection 객체를 생성하는 부분 -> 여기서 서버 오류가 발생
 - CP를 사용하여 HTTP 요청에 매번 위의 1-5의 단계를 거치지 않도록 처리
- CP의 활용
 - WAS가 실행되면서 미리 일정량의 DB Connection 객체를 생성하고 Pool 이라는 공간에 저장
 - HTTP 요청에 따라 필요할 때 Pool에서 Connection 객체를 가져다 쓰고 반환

Database 설정

- 라이브러리 설정(pom.xml)
 - DBCP(DB Connection Pool)

```
<!-- DB Connection Pool -->  
<dependency>  
  <groupId>org.apache.commons</groupId>  
  <artifactId>commons-dbcp2</artifactId>  
  <version>2.7.0</version>  
</dependency>
```

Database 설정

- MyBatis
 - 자바의 관계형 데이터 베이스 프로그래밍을 좀더 쉽게 할 수 있게 도와주는 개발 프레임워크
 - SQL문을 소스 코드로부터 완전 분리(자바 내부에서 쿼리문을 사용하지 않음)
- 구성
 - 환경설정파일(mybatis-conf.xml)
 - Mybatis 전반에 걸친 세팅
 - 필요 Artifact : mybatis, mybatis-spring
 - Mapping 설정파일
 - 사용할 SQL문들 정의

Database 설정

- MyBatis 라이브러리 설정(pom.xml)

```
<!-- MyBatis -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.5.9</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>2.0.6</version>
</dependency>
```

Database 설정

- MyBatis 설정(root-context.xml)

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation"
        value="classpath:com/****/****/dao/mybatis-conf.xml"/>
</bean>

<mybatis-spring:scan base-package="com.icia.movieinfo.dao"/>
```

Database 설정

- MyBatis 환경설정

- DAO(Data Access Object) 패키지를 생성하고 내부에 MyBatis 설정 파일(xml)과 DAO 인터페이스 생성 (**<mybatis-spring:scan>으로 Mybatis와 DAO 연동 처리**)
 - MyBatis와 DAO가 연동되어 SQL 쿼리문 실행
 - 설정 파일(xml)에 다음 문장을 추가

```
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
```

- 위 문장으로 MyBatis config 용 태그를 사용할 수 있음
- Bean 객체를 설정 파일에 지정(자료형에 긴 패키지 이름을 별칭으로 사용하도록 지정)

```
<configuration>
  <typeAliases>
    <typeAlias alias="별칭1" type="Bean 객체 패키지.클래스1명"/>
    <typeAlias alias="별칭2" type="Bean 객체 패키지.클래스2명"/>
    ....
  </typeAliases>
</configuration>
```

Database 설정

- MyBatis 환경설정
 - DTO class 생성
 - DB에 저장 또는 불러올 데이터 객체
 - DTO 패키지 생성 후 클래스 생성
 - @Alias("별칭") 어노테이션 사용
 - MyBatis 설정 파일과 동일한 별칭 사용(typeAlias의 alias 속성)
 - Mapper에 설정하는 자료형이 객체일 경우 패키지명을 생략하도록 만듦
 - 멤버 변수 생성 후 Setter/Getter 생성
 - MyBatis에서 쿼리를 실행하거나 쿼리 실행 후 결과값을 자동으로 해당 멤버변수에 주입할 때 Setter/Getter 메서드를 사용

Database 설정

- MyBatis Mapping 설정

- Mapper xml 파일 생성

- 사용할 SQL 쿼리문과 자바 코드의 연결용 설정 파일
 - mapper 패키지 생성 후 xml 문서 생성
 - 설정 파일(xml)에 다음 문장을 추가

```
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
```

- 위 문장으로 MyBatis mapper용 태그를 사용할 수 있음

```
<mapper namespace="DAO_Interface_패키지.인터페이스명">
    <select id="메서드명" resultType="반환형" parameterType="매개변수형">
        SQL 쿼리 문장
    </select>
</mapper>
```

- select, insert, update, delete 태그에 따라 쿼리문 작성 및 해당 메서드 입력

Database 설정

- MyBatis Mapping 설정
 - DAO 인터페이스 생성
 - Mapper xml의 namespace에 설정된 패키지에 생성(MyBatis 설정 파일과 같은 위치)
 - 인터페이스로 생성하는 이유
 - MyBatis가 쿼리문을 실행하여 결과의 반환까지 처리하기 때문에 메서드의 정의가 필요 없음
 - Mapper와 맞춰야 하는 내용
 - 메서드의 개수 = mapper xml의 요소 개수
 - 메서드의 반환형 = mapper xml의 resultType(wrapper class) : `int(메서드) = Integer(Mapper)`
 - 메서드의 매개변수 = mapper xml의 parameterType

Database 설정

- MyBatis Session 빌드 설정(root-context.xml)
 - dataSource, Transaction, sqlSessionFactory 설정
 - dataSource bean
 - DB 접속 정보 bean
 - sqlSessionFactory
 - sqlSessionFactory bean을 설정
 - MyBatis와 DB를 연결하는 객체
 - dataSource를 참조하여 MyBatis와 MySql을 연동시킴
 - DAO bean 등록
 - <mybatis-spring:scan base-package="프로젝트패키지.dao">

Database 설정

- dataSource bean
 - DB 접속에 필요한 정보 객체

```
<beans:bean id="dataSource"  
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <beans:property name="driverClassName" value="com.mysql.jdbc.Driver"/>  
    <beans:property name="url" value="jdbc:mysql://아이피:3306/DB명"/>  
    <beans:property name="username" value="root"/>  
    <beans:property name="password" value="DB암호"/>  
</beans:bean>
```

- class와 각 property name의 값은 위와 같이 작성 해야함(고정값)
 - driverClassName, url, username, password
 - 다르게 작성할 경우 문법 error 발생

Database 설정

- sqlSessionFactory bean
 - MyBatis가 dataSource와 mapper xml을 사용하여 DB와 연동

```
<beans:bean id="sqlSessionFactory"  
    class="org.mybatis.spring.SqlSessionFactoryBean">  
    <beans:property name="dataSource" ref="dataSource"/>  
    <beans:property name="configLocation"  
        value="classpath:com/****/****/dao/mybatis-conf.xml"/>  
</beans:bean>
```

- class는 위와 같이 작성 해야함(고정값)
 - name의 dataSource, configLocation(SqlSessionFactoryBean 클래스의 내부 변수이름, 고정값)
 - ref의 값은 dataSource bean에서 작성한 id 값과 동일

Controller

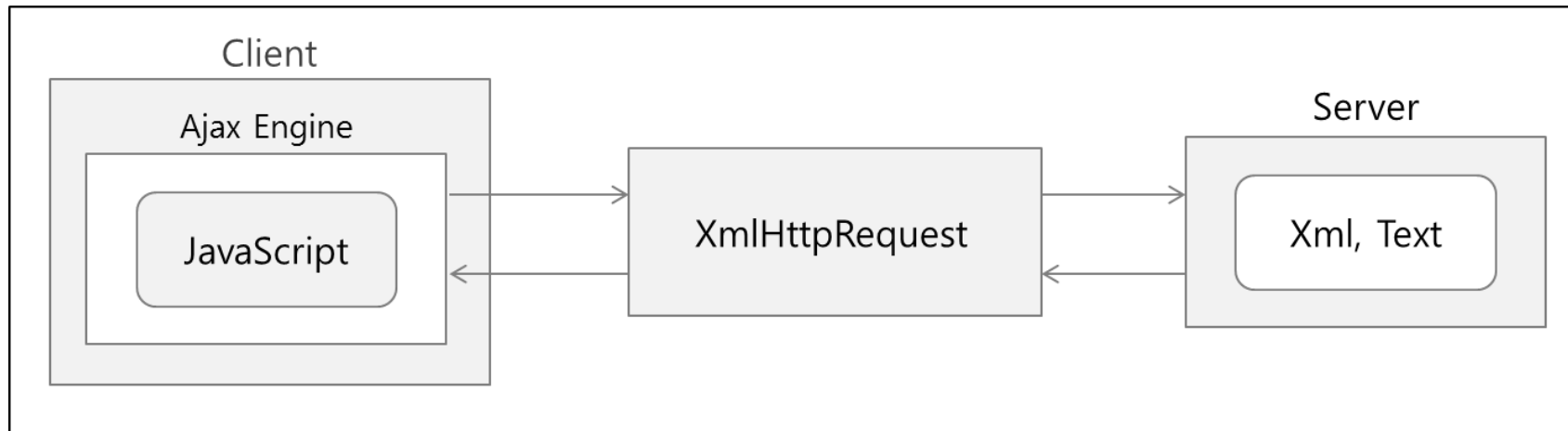
- 컨트롤러는 요청 URL에 대하여 서비스 메서드를 연결하는 작업으로 구성
 - 실질적인 처리는 서비스 객체에서 수행
- 처리 메서드 정의
 - @RequestMapping 사용 : @WebServlet과 같은 어노테이션으로 Controller에서 요청 URL과의 연결에 사용하는 어노테이션
 - value : 메서드와 연결할 호출 url
 - method : 전송 방식(GET/POST) 지정
 - method를 지정할 경우 지정된 전송 방식만 처리하는 메서드 작성
 - method를 지정하지 않을 경우 GET/POST 모두에 응답
 - @GetMapping, @PostMapping을 사용
 - 반환값의 설정
 - 전달할 데이터 없이 페이지 전환만 처리할 경우 : String으로 jsp 파일명만 반환
 - 전달할 데이터가 있을 경우 : Model/ModelAndView 객체를 선언하여 반환

Service class

- 의존 객체 작성
 - @Service 어노테이션을 클래스 앞에 붙임(클래스가 서비스용임을 알림)
 - DB 연동용 DAO 인터페이스 객체 : MyBatis 연결용 인터페이스
 - servlet-context.xml의 DAO bean의 id와 동일한 객체명을 지정
 - HttpSession : 세션을 사용하기 위한 객체
 - @Autowired 어노테이션 사용
 - 해당 클래스의 객체를 주입(인스턴스 가져오기) : new 또는 getInstance() 하지 않아도 자동으로 처리. 스프링 프레임워크에서 자동으로 클래스 이름을 검색하여 처리해 줌.
 - Model 또는 ModelAndView 객체
 - Model/ModelAndView : View로 데이터를 전달하기 위해 사용하는 객체
 - Model 객체는 데이터만 저장
 - ModelAndView 객체는 데이터와 데이터를 보여줄 view(jsp) 정보도 함께 저장
 - 서비스 처리용 메서드 정의

Ajax(Asynchronous JavaScript and XML)

- Web에서 화면을 갱신하지 않고 Server로부터 Data를 가져오는 방법을 제공
- 동작 원리
 - Browser에서 서버로 보낼 Data를 Ajax Engine을 통해 Server로 전송
 - Ajax Engine은 JavaScript를 통해 DOM을 사용하여 XMLHttpRequest 객체로 Data를 전달
 - Data를 전달 할 때 화면전체의 HTML을 전달하지 않고 Text 또는 Xml형식으로 Browser에 전달



Ajax(Asynchronous JavaScript and XML)

- 사용 방법
 - jQuery 문법을 사용하여 간단히 구현

```
$.ajax({  
    url : '요청 URL 주소'  
    [, Options]  
});
```

- URL은 필수 요소이므로 반드시 구현해야 하는 Property
- 다양한 속성들 중에서 필요한 Option을 선택해서 구현

Ajax(Asynchronous JavaScript and XML)

- Options

key	설명
url	요청이 전송되는 URL이 포함된 문자열 입니다.
type	HttpRequest방식 입니다. (Get/Post)
timeout	HttpRequest에 대한 제한 시간을 지정합니다.(단위 : ms)
success	HttpRequest 성공시 이벤트 핸들러 입니다.
error	HttpRequest 실패시 이벤트 핸들러 입니다.
complete	HttpRequest 완료시 이벤트 핸들러 입니다.
data	HttpRequest 후 return하는 값 입니다.
dataType	HttpRequest 후 return하는 데이터의 Type을 지정합니다. (xml,html,sript,json,jsonp,text)
async	요청시 동기 유무를 선택할 수 있습니다.(True/False)
cache	브라우저에 의해 요청되는 페이지를 캐시할 수 있습니다 (True/False)
beforeSend	HttpRequest 전에 발생하는 이벤트 핸들러 입니다.
global	전역함수 활성화 여부를 설정합니다. (True/False)

RedirectAttributes

- 리다이렉트가 발생하기 전에 모든 플래시 속성을 세션에 복사
- 리다이렉션 이후에는 저장된 플래시 속성을 세션에서 모델로 이동시킴.
 - 헤더에 파라미터를 붙이지 않기 때문에 URL에 노출되지 않음.
- addFlashAttribute()
 - RedirectAttributes가 제공하는 메소드.
 - 리다이렉트 직전 플래시에 저장하는 메소드다. 리다이렉트 이후에는 소멸한다.

